An Innovative Approach to an Isometric Game Engine

Authors: Caius Grozav, Dawn Mercer and Dmitriy Svetov, Seneca College, Toronto, ON

Abstract
In educational game design critical factors include a high level of engagement and play value along with ease of distribution. In the development of a game related to health issues under development as part of the SAGE project, it was determined that a Flash game engine could be designed to meet the increasingly sophisticated requirements of early teen gamers/learners as well as being readily available online for ease of use in educational settings. This paper explores the issues encountered in developing a Flash game engine and the unique solutions developed to date.

Introduction
In typical game design, the game story is imagined, the game play is defined and the internal elements are described before development begins. The constraints of the project defined here necessitated a creative process in game engine design. This involved an iterative process, based on trial and error, to formulate the functionality of a successful new game engine (El Rhalibi, 2005). As development began, the game concept was defined for the programming team along with some basic constraints of the game. The programming group was assigned to explore building an on-line engine that could accommodate open spaces with several customizable characters. Other members of the team worked in parallel to precisely define the game logic.

The game engine development described here is part of the SAGE Project. ("Simulation and Advanced Gaming Environments for Learning" is a SSHRC funded project exploring the potential of simulations and games to support learning in light of new technologies, new media and our knowledge of how people learn.) The game created in this project, Contagion, aims to provide an educational experience relating to health issues. It is expected to function in Canadian high schools and must run over the Internet, even at low bandwidth. Since the game will be played in homes and school computer labs, no component of the game should be installed on the client machine. Most important is that the game must be appealing to the target group, and provide the look and feel that approximates commercial 3D gaming environments.

Background Research

Feedback from our target user group indicated that:

- They preferred simulation games online;
- They do not like 2D environments for play;
- They do not like the boxy environment in isometric spaces (jumping from box to box);
- They want to be able to take full ownership of their character.

Existing online games were explored in depth. From sites such as Top 200 Hot Games http://www.colonize.com/content/displaygames.nc.php?a=51&b=100 it became clear that the game engine required for this project must allow the player to be more involved in the game play by having the main character/s take on the user point-of-view. The 'shoe box' perspective of games such as Pharaoh's Tomb http://www.groovyjava.com/games/pharaoh-tomb/pharaohs-tomb.php was not broad enough to provide the environment envisioned for this project.

Research confirmed that 'Tile Based Games' such as those demonstrated at the sites listed below would be explored as initial models for development.

- Tile Based Games http://www.tonypa.pri.ee/tbw/
- Tile / Object Based Flash Tutorial http://www.strille.net/tutorials/part1_scrolling.php
- Moxiecode http://oos.moxiecode.com/

Brief Survey of Game Engine Development

Flash was designated as the authoring tool, and an isometric environment was accepted as a compromise between the flat 2D and full 3D. Since no existing Flash game engine

could be identified that would fulfill the requirements of the game design, a new game engine would have to be created.

Several technical solutions were tested in order to minimize the load on the client and thus increase the speed of game play.

.

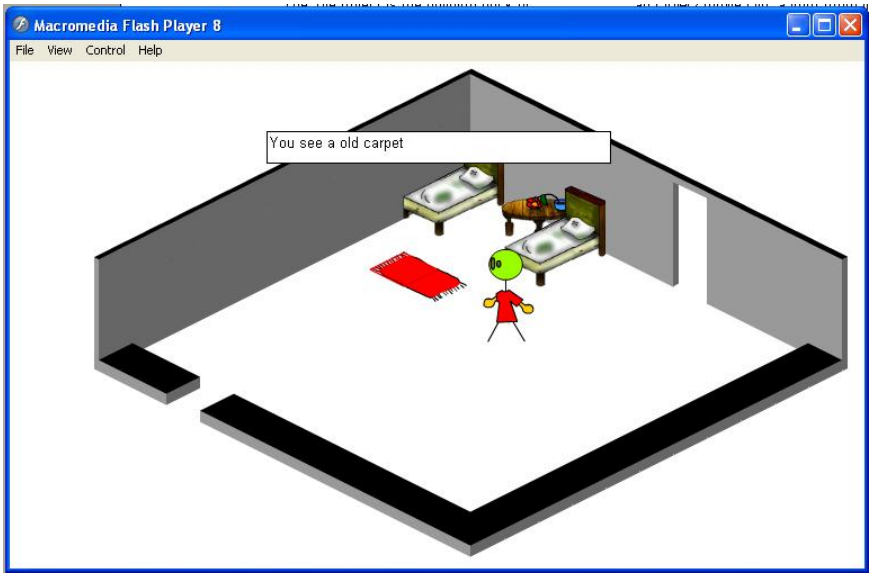


Fig. 1 - Earliest stage of game development.

Figure 1 illustrates an early tile-based environment that allows the user to control the movement of the character on the screen and the visual layout of the space. The character moves on the screen from tile to tile (although tiles are not visually explicit). The “Shoe Box” layout was the major problem in this model because wider environments where required to accommodate by the narrative.

Fig. 2 – Eliminating the “free” space.

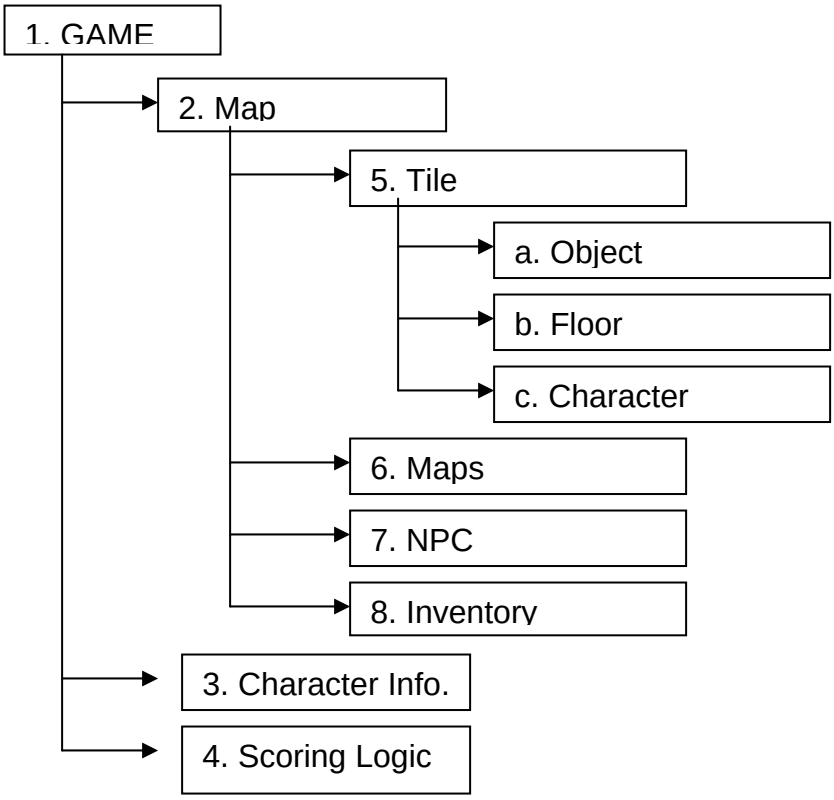
In the second model tiles are defined as Flash movie clips. The space is opened up by adding more tiles. The movement is simulated by loading and removing tiles dynamically (see Fig. 2). This resulted in an overload of the CPU. The engine was unacceptably slow; cropping of graphics in these step-sized tiles was also a problem.

Fig. 3 - Large tiles

We experimented with models of various tile sizes trying to find the best compromise for tile size and speed (Fig. 3). The engine was now fast enough, but the character could not get close to objects on these large tiles that were required to provide adequate game speed. This problem was solved by subdividing tiles into invisible squares ("nonwalkable tiles") dynamically and redesigning the logic of the movement as detailed in the following discussion.

Components of the GAME ENGINE

The basic structure of our game engine is shown in the following diagram:



Game component descriptions:

1. GAME
Contains all preloaders for the components and initial settings for the current level.

2. Map
The Map is an array of tile objects – the class defines the movement and general behavior of the tiles on the screen (movement is simulated by the map behind the character in response to mouse click or arrow key press). It also contains the mapping of the isometric spaces (Maps) and the non playable characters (NPC's).

3. Character Info.
This class holds information about the type of character, gender, and color/appearance of character components. It has its own database connectivity functions, allowing saving and/or retrieving character information.

4. Scoring Logic
This is a set of classes, defining the scoring logic for every level in the game. In our model we have 3 levels, but more can be added by adding new classes.

5. Tile
The Tile object is the building block of the isometric environment. It is designed as a stand-alone Flash movie clip, and it is preloaded every time the environment changes.

a. Object
The *Object* is a flash movie clip that is part of every *Tile* instance. All possible layouts for a tile in a given environment are defined in key frames (some objects are stored in more than one frame – e.g. a sofa needs 2 tiles, and a van 3). Objects are designed in 3D Studio MAX and/or Swift 3D, cropped to the size of a tile, and exported to Flash.
Defining an environment implies building an *Object* movie clip, a floor (both inside a *Tile* object), and a *Maps* array defining which key frame of the *Object* movie will be displayed on each *Tile* instance.
For 'nonwalkable' tiles the key frames of the *Object* movie clip also contain an array of codes that detail the characteristics of the current visual

layout as later explained in the section on Movement and Obstacle Avoidance.

b. Floor

The floor is a movie clip instance inside the *Tile* object, defining the visual layout (texture) of the floor in a given environment; it also detects the presence of the character movie clip on the tile.

c. Character

Every character in the game is designed as a standalone movie clip and is loaded on top of the current tile in the environment. Characters are designed and animated in 3D Studio MAX and/or Swift 3D. For each character, 5 directions of movement are rendered (the remaining 3 directions are displayed by mirroring). The animated character is then exported for Flash on layers (highlights, colors, and shadows). In Flash, the colors are separated in layers corresponding to the customizable components of the character and converted into color objects.
The actual color of every component can be customized in the character customization section and is stored in a database by the Save command.

6. Maps

The Maps are arrays of codes controlling how the space should be rendered. The size of the array defines the number of tiles in the environment (in our case, 10x10 smallest, and 30x30 maximum). The custom codes in the array elements define the visual appearance of the corresponding tile and its type. There are three basic types of tiles defined: “walkable” tiles (the character is free to move anywhereon the tile), “nonwalkable” tiles (there is an object on the tile obstructing some movements of the character), “pickable” tiles (there is an object that can be picked-up/engaged by the character on the tile), and any combination of the above mentioned.

7. NPC (Non Playable Characters)

Non Playable Characters (NPCs) are loaded on top of the map, but they are not part of it – this way they can be loaded and unloaded as needed, without affecting the playable space of the game. The player can interact with them, but cannot control them.

Two types of NPC’s are defined:
Static – they occupy a fixed position on the map and move with it (e.g. a sick person to be treated by the player).
Dynamic – they are rendered the same way the character is, and they move on top of the map, following an invisible grid of nodes and arches at random (like Pac-Man monsters). The grid moves with the map, its logic is totally independent, but programmable if needed.

8. Inventory

Inventory is a separate class organized as an array of objects that allows assigning items not only to the character and NPC’s, but also to objects (desk, cupboard, bin). These items can be used by the player (e.g. a mask can be moved from the inventory of the cupboard to the inventory of the character).

Unique Approaches to Movement and Obstacle Avoidance Developed

a. Large Tiles

An array of 7x7 tiles is displayed from the game environment described by information in the *Maps*.

Fig. 4

The tile can have its own actions, an “object” movie clip, and a “floor” movie clip (Fig. 4).

Fig. 5

The “floor” movie clip contains all possible graphic layouts of the floors in one level stored in key-frames. It also has actions to determine the presence of the character on the tile.

Fig.6

All the objects and graphics (other than floors) in an environment are loaded as key-frames in the "object" movie (Fig. 6). On the screen we display an array of 49 instances of the large tile, each one displaying one floor key-frame and one object key-frame according to the information stored in the map of the environment.

b. Movement

The character will always be positioned in the center of the "stage". Movement is simulated by moving the map behind it. The character appears to walk by looping one of the 8 possible directions of movement in the isometric space.

i. Movement between tiles

At the level of one tile, the map is actually moved behind the character. In order to avoid continuous loading and unloading *Tile* objects, when the character is moving onto a different tile, the entire environment is rebuild. The *Object* and *Floor* movie clips inside every *Tile* instance are reset to display different key frames and the entire map is repositioned before being displayed. As a result, the Map moves only the size of an isometric tile, and repainting the environment creates the illusion of continuous movement. The relative position of the character in the repainted environment is reflected by changing the depth of the *Character* movie clip in conjunction with the display of objects on neighboring tiles.

ii. Obstacle avoidance

In order to increase the realism of the simulation in achieving obstacle avoidance the following new approach was developed:

On arrow key press, the character will take one step in the corresponding direction. If the mouse button is clicked on the "stage", the coordinates of that point will be stored as the destination. Distances on x and y between the current position and the destination are computed, and based on those values, the character starts walking, always selecting the shortest path (with the constraint of only 8 possible directions of movement).

Three basic types of tiles are defined:

- "Walkable" tiles have no restrictions in the movement of the character in any direction.
- "Nonwakable" tiles have objects that restrict the movement of the character, totally or just in some areas.
- "Pickable" tiles have an associated inventory (objects that the player can interact with).

If a tile is labeled as nonwalkable, when the character steps on it a function is invoked subdividing the tile into invisible squares, the size of a step. On such a tile the character will always take a real step and attempt a virtual one in the direction desired by the player. If the virtual step is not possible, because an object occupies the next invisible square, the character will attempt a second virtual step in one of the neighboring directions (at random). If this second step is possible, movement will continue otherwise the character stops in front of the object. The key frame defining a nonwalkable tile contains an array describing the invisible squares blocked by the object on the tile. The array is generated by a separate Flash application when the space is designed, based on the shape of the graphic object on the tile.

c. Character and NPC's

The Character and the Non Playable Characters (NPC's) movie clips have exactly the same structure and are sharing the same basic code except for their logic of movement, which is included from different Action Script (*as*) files. The visual appearance of a character is also designed as an external movie (*.swf*) and is 'created' by the player during the character customization dialogue, or specified in the game logic, for NPC's. Characters are designed and animated in 3D Studio Max and/or Swift 3D, then rendered and exported on layers as frame-by-frame animations for Flash (*.swt* – highlights, colors, and shadows). The animation contains 5 sequences of one step identified by labels. The remaining 3 steps needed to cover the 8 possible directions of movement are created by mirroring the character (scaleX = -100%). After import, the color layer is again broken into layers corresponding to the components that can have custom colors. Shapes in each layer are then converted to color objects and custom named in order to make them visible for the character customization dialogue. As

specified above, the movement logic is assigned using the *include* directive from external as files. It is predefined for the character only, and can be customized for NPC's.

d. Tiles as external movies
The tile is an external *.swf* loaded by the *Map* and controlled by information stored in the *Maps* array(s). Building a new level in the game, or a different game, involves the following steps:

- Design the environment, crop to the large tile size, and load components in the structure of the tile object;
- Fill in a Maps array containing the arrangement of tiles and their attributes (for nonwalkable tiles the invisible squares array must be built using the existing Flash app);
- Create the graphics for Characters and NPC's;
- Define custom movement logic and behavior for NPC's;
- Define scoring logic in external action script files (*as*) and any additional items that should be loaded during the game as external movies (*.swf*);
- Define inventories and their visual appearance as external movies (*.swf*).

Although not yet a fully developed game engine, this approach proved elastic enough to accommodate different spaces and scenarios in the Contagion game, at both character and mission level. The programming effort in game development was also reduced to less than 30%.

For the future

We hope that the newly released version of Flash will improve the performance of the engine and that new features will allow us to improve the quality of graphics in the game.
A recursive "smart-mouse" logic for more sophisticated obstacle avoidance was tested, at both map and "stage" level; it resulted in an unacceptable overload for the client machine. A recursive method at the tile level, based on the invisible squares, is under development. We are also considering the alternative of a custom optimized web-service that could serve back to the character the best sequence of steps.

References

Avery, A. (2005). "Beyond P-1: Who plays online" [Electronic version]. Retrieved: August 15, 2005, from http://www.gamesconference.org/digra2005/viewabstract.php?id=155

El Rhalibi, A., England, D., Hanneghan, M., Tang, S. (2005). "Extending software models to game design" [Electronic version]. Retrieved August 15, 2005, from http://www.gamesconference.org/digra2005/viewabstract.php?id=380

Klas, (2002). "Tilebased games in Flash 5". Retrieved May 2004 from http://oos.moxiecode.com/

"Top 200 Hot Games". Retrieved May 2004, from http://www.colonize.com/content/displaygames.nc.php?a=51&b=100

Pa, T. (2004). "Tile Based Games". Retrieved May 2004, from http://www.tonypa.pri.ee/tbw/

"Pharaoh's Tomb". Retrieved May 2004 from http://www.groovyjava.com/games/pharaoh-tomb/pharaohs-tomb.php

Rouse, R. (2001). "Game Design Theory & Practice", Wordware Publishing Inc.

Stridsman, M. (2004). "Tile based / object based Flash tutorial". Retrieved May 2004, from http://www.strille.net/tutorials/part1_scrolling.php

Acknowledgements

The creativity and enthusiasm of Co-op students from the Seneca College Computer Studies Department, Rita Kassab and Mikko Haapoja, contributed greatly to the outcome of this project. The authors also wish to acknowledge the critically important role of other members of the Contagion development team including Suzanne de Castell, Jennifer Jensen, Nicholas Taylor and Nis Bojin who provided innovative suggestions and critiques that helped to move the game engine to the current level of functionality.